

Phân tích và thiết kế thuật toán

🕒 Created	@January 3, 2025 1:12 PM
☑ Reviewed	<input type="checkbox"/>

Môn học: Phân tích và Thiết kế Thuật toán

Tỉ lệ điểm: 4/6

Cấu trúc môn học

- Chương 1: Tổng quan
- Chương 2: Tìm kiếm
- Chương 3: Chia để trị
- Chương 4: Quay lui
- Chương 5: Tham lam
- Chương 6: Quy hoạch động

Tài liệu học tập

- B-learning
- Giải thuật và lập trình - Lê Minh Hoàng
- Nhập môn thuật toán - MIT

Chương 1: Tổng quan

1. Thuật toán

- Mục tiêu: Thuật toán và Code (C/C++)
- Môi trường thực hành: laptrinhonline.club
- Khái niệm của thuật toán: là một dãy các thao tác biến đổi dữ liệu, biến đổi thông tin dữ liệu đầu vào (INPUT) thành thông tin dữ liệu đầu ra (OUTPUT) sau hữu hạn bước.

VD: Biến đổi thông tin thành thông tin

- Thông tin : Hiểu nôm na là thông tin về 1 sự vật hiện tượng từ không biết thành biết, từ biết ít thành biết nhiều.

Tính đầy đủ và đúng đắn của logic:

Thuật toán Euclid

Cho a và b nguyên dương. Tìm ước chung lớn nhất ?

- Muốn tìm ước chung lớn nhất của a và b thì chính là ước chung lớn nhất của a-b và b
- Biểu diễn : $(a,b) = (a-b,b) = d$
- VD: $(27,72) \rightarrow (27,45) \rightarrow (27,18) \rightarrow (9,18) \rightarrow (9,9) \rightarrow 9$ là UCLN của $(27,72)$
- Thuật toán:
 - B1: Nhập a và b

- B2: Kiểm tra $a \neq b$
 - Nếu $a > b$ thì $a = a - b$
 - Ngược lại $b = b - a$
 - Quay lại B2
- B3: Xuất kết quả
- CODE

```
#include <bits/stdc++.h>
using namespace std;
main() {
    //HS LAM
    int a,b;
    cin >> a >> b;
    while(a!=b) {
        if (a>b) {
            a = a-b;
        } else {
            b = b-a;
        }
    }
    cout << a;
    //THAY CODE
    B1:
        int a,b;
        cout << "a = ";cin >> a;
        cout << "b = ";cin >> b;
    B2:
        if (a-b) {
            a>b?a-=b:b-=a;
            goto B2;
        }
    B3:
        cout << "UCLN " << a;
}
```

Máy sinh ngôn ngữ Automata

```
#include <bits/stdc++.h>
using namespace std;

main() {
    char x[100000],*p = x;
    scanf("%s",x); //gets(x); → getline(cin,x);

    D:
        if (*p == ' ') {p++;goto D;}
        if (*p==0) goto K;
        cout << char(toupper(*p++)); goto T;
    T:
        if (*p == ' ') {p++;cout<<" ";goto D;}
}
```

```

    if (*p==0) goto K;
    cout << char(tolower(*p++)); goto T;
K:
    return 0;
}

```

2. Đặc trưng thuật toán

2.1. Tính xác định

- Mỗi thuật toán gắn với 1 bài toán đặc trưng bởi đầu vào, đầu ra và điều kiện của chúng.

→ VD: TÌM NGHIỆM BẬC 2

- $ax^2 + bx + c = 0$ (Dùng Viet)
- Nếu không dùng Viet \Rightarrow Dùng Rôdu (Chỉ áp dụng cho số nguyên)
 - $ax^2 + bx = -c$ (x phải là nghiệm của c) \Rightarrow x sẽ là ước của c;

Note: Sắp xếp mà dựa vào so sánh thì nhanh nhất cũng chỉ là $O(\log n)$

→ VD: SẮP XẾP KÍ TỰ TRONG XÂU

- dhgtvt \rightarrow dhjttv
- TT: Counting Sort
 - B1: Nhập xâu x
 - B2: Đếm tần xuất 'a' \rightarrow 'b' \rightarrow ?
 - B3: Duyệt các kí tự từ 'a' \rightarrow 'z' theo
- CODE

```

#include <bits/stdc++.h>
using namespace std;

main() {
    char x[10000], *q=x;
    scanf("%s", x);
    int d[126] = {}; // 'a'  $\rightarrow$  97; 'z'  $\rightarrow$  122;
    for (char *p = x; *p != '\0'; p++) d[*p]++;
    for (char c = 'a'; c <= 'z'; c++) {
        while(d[c]--) *q++ = c;
    }
    printf("%s", x);
}

```

2.2. Tính dừng

- Thuật toán chỉ dừng được nếu nó dừng sau hữu hạn bước

→ VD: CHẶT NHỊ PHÂN

BT: Cho các điểm A, B, M. Tìm điểm nằm trên đoạn AB gần M nhất

TT:

- B1: Nhập A, B;

- B2: $\epsilon = 1e-4$
- B3: Kiểm tra $|Ax - Bx| > \epsilon$ hoặc $|Ay - By| > \epsilon$
 - $C = ((Ax + Bx)/2, (Ay + By)/2)$
 - Nếu $MA > MB$ thì $A = C$ ngược lại $B = C$
 - Quay lại B3
- B4: Xuất A

CODE

```
#include <bits/stdc++.h>
using namespace std;

typedef pair<double,double> Diem;
#define x first
#define y second

double bpkc(Diem A, Diem B) {
    return (A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y);
}

main() {
    Diem A,B,C,M;
    double e = 1e-4;
    cin>>A.x>>A.y>>B.x>>B.y>>M.x>>M.y;
    while(abs(A.x-B.x) > e || abs(A.y-B.y) > e) {
        C = {(A.x + B.x)/2, (A.y+B.y)/2};
        (bpkc(A,M) > bpkc(B,M)) ? A=C : B=C;
    }
    cout << setprecision(4) << fixed << A.x << " " << A.y;
}
```

BTVN: 1.Tìm điểm trong tam giác và gần M nhất

2.Tìm điểm trong tam giác

3. Tìm điểm trong hình tròn

2.3.Tính tuần tự

- Mỗi thuật toán có nhiều bước theo 1 trình tự bước trước thực hiện trước bước sau thực hiện sau

→ VD: THUẬT TOÁN Sàng ERATOTSTHENES

```
#include <bits/stdc++.h>
using namespace std;
#define int long
main() {
    int n;cin>>n;
    bool s[n+1];
    fill(s,s+n+1,true);
    for (int i = 2;i<=n;i++) {
        if (s[i]) {
            cout << i << " ";
            for (int j = i*i;j<=n;j+=i) s[j] = false;
        }
    }
```

```
}  
}
```

BTVN: Giả thiết Goldbach

2.4. Tính phổ dụng

- Một thuật toán có thể áp dụng cho nhiều bài toán.

→ VD: CÁI TÚI

Cho n đồ vật kích thước là a_1, a_2, \dots, a_n và một cái túi có kích thước là M

Nhiệm vụ của bạn là hãy lựa chọn các đồ vật xếp vào túi cho hợp lý sao cho tổng kích thước được chọn không vượt quá kích thước túi mà nhưng phải lớn nhất có thể

- Input**

Dòng đầu gồm hai số nguyên dương là số đồ vật $n (1 \leq n \leq 20)$ và kích thước túi $M ((1 \leq M \leq 1000))$

Dòng cuối là n số nguyên dương tương ứng với kích thước các đồ vật có giá trị không vượt quá 1000

- Output**

Một số tự nhiên là kích thước thu được lớn nhất có thể của cái túi, tất nhiên nếu không xếp được đồ vật nào thì xuất ra 0

Input

```
4 40  
17 26 19 8
```

Output

```
36
```

Giải thích Ta chọn 2 đồ vật kích thước là 17 và 19 có tổng là 34

CODE:

```
#include <bits/stdc++.h>  
using namespace std;  
  
main() {  
    int n,m,res=0;cin>>n>>m;  
    int x[n+5] = {},a[n+5];  
    for (int i = 1;i<=n;i++) cin >> a[i];  
    while(x[0]==0) {  
        int T=0;  
        for (int i = 1;i<=n;i++) T+=a[i]*x[i];  
        if (T<=m) {  
            res = max(T,res);  
        }  
        int j ;  
        for (j = n;x[j]==1;j--) {  
            x[j] = 0;  
        }  
        x[j] = 1;  
    }  
    cout << res;
```

```
}
```

→ VD: SINH CÁC DÃY NHỊ PHÂN

Nhập vào số nguyên dương n ($1 \leq n \leq 9$), hãy sinh các dãy nhị phân có độ dài n .

- **Input**

Số nguyên duy nhất n

- **Output**

Mỗi dòng một dãy nhị phân được liệt kê theo thứ tự từ điển.

- **Example**

Input:

```
3
```

Output:

```
000
001
010
011
100
101
110
111
```

```
#include <bits/stdc++.h>
using namespace std;

main() {
    int n ;cin>>n;
    int x[n+5] ={};
    while(x[0]==0) {
        for (int i = 1;i<=n;i++) cout << x[i] << " ";
        cout << "\n";
        int j ;
        for (j = n;x[j]==1;j--) {
            x[j] = 0;
        }
        x[j] = 1;
    }
}
```

2.5.Tính tối ưu

- Mỗi bài tập có cách giải → luôn có cách tốt hơn

→ VD: ROBOT LĂN SƠN (ROBOT QUÉT VÔI VER3)

Năm 2018, Khoa Điện - Điện Tử của Trường Đại học Giao thông Vận tải kỉ niệm 20 năm thành lập. Các bạn sinh viên đã nghiên cứu làm ra Robot có khả năng lăn sơn tường. Robot được cho thử nghiệm quét vôi bức tường chạy dài m mét được đánh số từ vị trí 0 cho đến vị trí m. Robot được chạy thử nghiệm n lần, lần thứ i Robot sẽ quét vôi từ vị trí ai đến vị trí bi có độ dài là bi-ai.

Sau các lần thử nghiệm ở những chỗ nào được lăn sơn ít nhất k lần thì chỗ đó coi như đã sơn xong, các bạn sinh viên muốn biết là bức tường đã sơn xong bao nhiêu mét, biết rằng các lần lăn sơn có thể phủ lên nhau.

- **Input**

Dòng đầu gồm ba số nguyên dương n, m và k tương ứng với số lần quét và độ dài bức tường và số lần sơn ít nhất thì coi như đã sơn xong $1 \leq n \leq 106, 1 \leq m \leq 106, 1 \leq k \leq n$.

n dòng tiếp theo mỗi dòng gồm hai giá trị ai, bi ($0 \leq ai < bi \leq m$).

- **Output**

Một số nguyên duy nhất là số mét mà đã được sơn ít nhất k lần.

- **Ví dụ 1**

Input

```
3 100 2
55 72
12 44
30 81
```

Output

```
31
```

Giải thích : Tường đã sơn ít nhất hai lần $30 \rightarrow 44$ và $55 \rightarrow 72$ tổng cộng là $(44-30)+(72-55)=31$ mét

- **Ví dụ 2**

Input

```
3 100 1
30 40
50 60
10 20
```

Output

```
30
```

- **Chú ý:**

Thuật toán rất chặt về thời gian, bạn hãy sử dụng *scanf* và *printf* thay vì *cin* và *cout*

- Tạo ra 1 mảng toàn 0 mỗi lần lăn sơn thì ++ và so sánh với k yêu cầu
- → Tối ưu hóa Cuốn chiếu

```
#include <bits/stdc++.h>
using namespace std;
main() {
    int n,m,k;cin>>n>>m>>k;
    int res = 0;
    int a[m]={};
```

```

while(n-->0) {
    int L,R;
    cin >> L >> R;
    a[L]++;
    a[R]--;
}
partial_sum(a,a+m,a);
for (auto x:a) {
    if(x>=k){
        res++;
    }
}
cout << res;
}

```

- **BTVN: Truy vấn tổng**

→ VD: LƯỢNG NƯỚC

Nhà mình làm lại cái nhà

Mua những cái cọc đóng tra làm nền

Mấy ngày mưa ướt liên miên

Hỏi bao nhiêu nước đang trên nền nhà.

Cho n cái cọc bê tông được đóng thành một hàng dài sát nhau, khi mưa rơi xuống thì nước mưa trên các cọc thấp sẽ còn đọng lại nếu có những cọc cao ngăn cho, giả sử bề ngang của mỗi cái cọc là một đơn vị hãy tính lượng nước đọng lại trên các cọc này.

- **Input**

Dòng đầu chứa số nguyên dương n có giá trị không vượt quá 106.

Dòng sau là n số tự nhiên đại diện cho chiều cao của n cái cọc đứng lẫn lộn sát nhau đảm bảo đọng nước ở đó không thoát được có giá trị không vượt quá 32768.

- **Output**

Một số tự nhiên là tổng số nước đọng lại trên tất cả các cọc biết rằng trời mưa mấy ngày qua đảm bảo lượng nước vô hạn đã rót xuống.

- **Ví dụ**

Input:

```

7
3 7 2 1 5 2 4

```

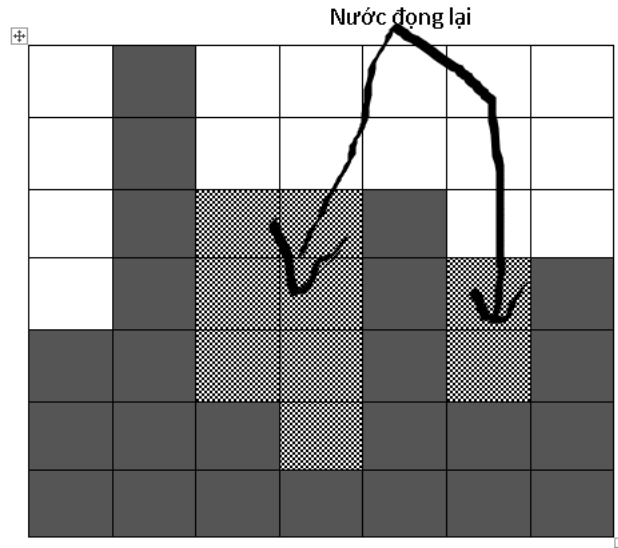
Output:

```

9

```

Giải thích: Lượng nước đọng lại ở cột 3 cột 4 thành một vùng và cột 6 như hình vẽ



```
#include <bits/stdc++.h>
using namespace std;
long myfunc(long a,long b) {
    return a>b?a:b;
}
main() {
    long n , res = 0;
    cin >> n;
    vector<long> a(n),L(n),R(n);
    for (auto &x:a) cin >> x;
    partial_sum(a.begin(),a.end(),L.begin(),myfunc);
    partial_sum(a.rbegin(),a.rend(),R.rbegin(),myfunc);

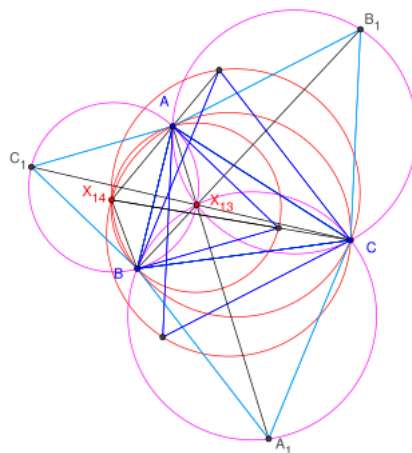
    for (int i = 1;i<n-1;i++) {
        res+=max(min(L[i-1],R[i+1])-a[i],0L);
    }
    cout << res;
}
```

2.6.Hình thức hóa

- Thuật toán muốn dùng được chỉ khi biểu diễn được bằng ngôn ngữ hình thức
- VD : Thuật toán leo đồi
- Cho n điểm trên xOy tìm diện tích nhỏ nhất chứa tất các điểm

→ BT: ĐIỂM FERMAT

Điểm Fermat là một điểm thú vị trong hình học phẳng, Toto học thầy Tichpx dạy đến tìm điểm Fermat bằng **thuật toán leo đồi (Hill Climbing Search)** liền bắt tay làm ngay nhưng chưa biết code như thế nào. Các bạn giúp Toto nhé.



Bài toán: Nhập vào tọa độ ba điểm trên mặt phẳng, tìm một điểm mà tổng khoảng cách của nó tới ba điểm đã cho là nhỏ nhất.

- **Input**

Ba dòng mỗi dòng chứa hai số thực có giá trị tuyệt đối không quá 1000 tương ứng là tọa độ trên mặt phẳng hai chiều Đề-Các (các điểm có thể trùng nhau).

- **Output**

Tọa độ điểm cần tìm có tổng khoảng cách đến ba điểm đã cho là ngắn nhất với sai số ít nhất ba chữ số sau dấu chấm thập phân.

- **Ví dụ 1**

Input

```
0.0 3.0
0.0 5.0
0.0 9.0
```

Output

```
0.000 5.000
```

- **Ví dụ 2**

Input

```
-3 0
0 7
3 0
```

Output

```
0.000000 1.732051
```

- Thuật toán
- Cho tam giác ABC tìm M để $MA + MB + MC \rightarrow \min$

Bước 1: Nhập tọa độ 3 điểm $A(x_A, y_A)$, $B(x_B, y_B)$, $C(x_C, y_C)$

Bước 2: Khởi tạo điểm M là trọng tâm: $M = \left(\frac{x_A + x_B + x_C}{3}, \frac{y_A + y_B + y_C}{3} \right)$

Bước 3: Đặt bước nhảy ban đầu $d = 10$

Bước 4: Trong khi $d > 10^{-5}$ thì: Duyệt các điểm N lân cận của M với bước d .
 Nếu tồn tại N sao cho $NA + NB + NC < MA + MB + MC$
 thì gán $M = N$. Nếu không có điểm tốt hơn thì $d = \frac{d}{2}$.

Bước 5: Xuất tọa độ điểm M (gần đúng)

Công thức khoảng cách: Giữa 2 điểm $P(x_1, y_1)$, $Q(x_2, y_2)$:
 $\text{dist}(P, Q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

```
#include <bits/stdc++.h>
using namespace std;

typedef pair<double, double> diem;
#define x first
#define y second

double kc(diem A, diem B) {
    A.x -= B.x; A.y -= B.y;
    return sqrt(A.x * A.x + A.y * A.y);
}

int main() {
    diem A, B, C, M;
    cin >> A.x >> A.y >> B.x >> B.y >> C.x >> C.y;
    M = {(A.x + B.x + C.x) / 3, (A.y + B.y + C.y) / 3};

    double eps = 1e-4, k = kc(A, M) + kc(B, M) + kc(C, M), d = 10;

    while (d > eps) {
        diem Next[] = {
            {M.x - d, M.y},
            {M.x + d, M.y},
            {M.x, M.y - d},
            {M.x, M.y + d}
        };
        for (diem N : Next) {
            double z = kc(A, N) + kc(B, N) + kc(C, N);
            if (z < k) {
                k = z;
                M = N;
                d /= 2;
                break;
            }
        }
        d /= 2;
    }

    cout << M.x << " " << M.y;
```

```
return 0;
}
```

3. Đánh giá độ phức tạp thuật toán

3.1. Khái niệm độ phức tạp thuật toán

- **Độ phức tạp** là lượng tài nguyên mà **thuật toán sử dụng** để giải bài toán.

Gồm 2 loại chính:

- **Độ phức tạp không gian (Space Complexity):**
→ Số bit (hoặc byte) bộ nhớ cần để lưu thông tin đầu vào, đầu ra, và dữ liệu trung gian.
- **Độ phức tạp thời gian (Time Complexity):**
→ Số phép toán mà thuật toán phải thực hiện, phụ thuộc vào:
 - Kích thước input (n)
 - Giá trị input

👉 Có 2 cách đánh giá:

- **Thực nghiệm:** chạy thử, đo thời gian, bộ nhớ
- **Lý thuyết:** phân tích thuật toán dựa trên số bước

3.2. Các ký hiệu độ phức tạp

→ a. Cận trên – Ký hiệu O (Big-O)

- Dùng để mô tả *giới hạn trên* cho độ phức tạp
- Công thức: $O(f(n)) = \{g(n) \mid \exists c > 0, \exists n_0 \text{ sao cho } g(n) \leq c \cdot f(n), \forall n \geq n_0\}$

$$O(f(n)) = \{g(n) \mid \exists c > 0, \exists n_0 \text{ sao cho } g(n) \leq c \cdot f(n), \forall n \geq n_0\}$$

- Ví dụ:

$$2n^3 + n + 7 = O(n^3) \quad 2n^3 + n + 7 = O(n^3) \quad 2n^3 + n + 7 = O(n^3)$$

→ b. Cận dưới – Ký hiệu Ω (Omega)

- Dùng để mô tả *giới hạn dưới* cho độ phức tạp
- Công thức: $\Omega(f(n)) = \{g(n) \mid \exists c > 0, \exists n_0 \text{ sao cho } g(n) \geq c \cdot f(n), \forall n \geq n_0\}$

$$\Omega(f(n)) = \{g(n) \mid \exists c > 0, \exists n_0 \text{ sao cho } g(n) \geq c \cdot f(n), \forall n \geq n_0\}$$

→ c. Cận chặt – Ký hiệu Θ (Theta)

- Dùng để mô tả độ phức tạp *chính xác* (cận trên và dưới trùng nhau)
- Công thức: $\Theta(f(n)) = \{g(n) \mid \exists c_1, c_2 > 0, \exists n_0 \text{ sao cho } c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n), \forall n \geq n_0\}$

$$\Theta(f(n)) = \{g(n) \mid \exists c_1, c_2 > 0, \exists n_0 \text{ sao cho } c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n), \forall n \geq n_0\}$$

- Tính chất: phản xạ

$$\begin{aligned} O(f(n)) &= f(n) \\ \Omega(f(n)) &= f(n) \\ \Theta(f(n)) &= f(n) \end{aligned}$$

- Tính chất: đối xứng

$$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$$

$$\text{Nếu } c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n) \Rightarrow f(n) = \Theta(g(n))$$

- Tính chất: chuyển đổi

$$f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$$

- Tính chất: bắc cầu

$$f(n) = O(g(n)) \text{ và } g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$

$$f(n) = \Omega(g(n)) \text{ và } g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$$

$$f(n) = \Theta(g(n)) \text{ và } g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$$

- Đánh giá độ phức tạp

- Thuật toán lặp

- QT cộng: Nếu thuật toán gồm 2 phần **A** và **B** thực hiện nối tiếp nhau (không lồng), thì tổng thời gian bằng tổng từng phần.
- Độ phức tạp sẽ lấy phần có tốc độ **chậm nhất**

$$T(n) = T_A(n) + T_B(n) = \max(T_A(n), T_B(n))$$

◆ Ví dụ:

- A: vòng lặp chạy $O(n)$
- B: vòng lặp chạy $O(n^2)$
- ⇒ Tổng độ phức tạp:

$$T(n) = O(n) + O(n^2) = O(n^2)$$

- QT nhân: Nếu thuật toán A gọi thuật toán B bên trong (lồng nhau), thì thời gian là **tích số lần chạy**.

$$T(n) = T_A(n) \cdot T_B(n)$$

◆ Ví dụ:

- A: vòng lặp chạy n lần
- B: mỗi lần chạy mất $O(n)$
- ⇒ Tổng độ phức tạp:

$$T(n) = O(n) \cdot O(n) = O(n^2)$$

- THUẬT TOÁN ĐỆ QUY

- Tính n!

```
long gt(int n) {
    return n ? n * gt(n-1) : 1;
}
```

- Đặt $T(n)$ là đpttt ta có

- $T(n) = C \quad (n=0)$
- $T(n-1) + C \quad (n>0)$

Ta có:

$$T(n) = T(n-1) + C = T(n-2) + 2C = \dots = T(0) + nC = C_0 + nC = \Theta(n)$$

- o Tính x^n

```
double Pow(double x, int n) {  
    return n ? pow(x,n-1):1;  
}
```

- o Đặt $T(n)$ là độ phức tạp thuật toán

Ta có:

$$\begin{aligned} T(n) &= C_0 \quad (n = 0, 1) \\ T(n) &= T(n-1) + C \quad (n > 0) \\ T(n) &= T(n-1) + C = T(n-2) + 2C = \dots = C_0 + nC = \Theta(n) \end{aligned}$$

- o Tính x^n

$$f(n) = \begin{cases} 1, & n = 0, \\ x^{\left(\frac{n}{2}\right)^2}, & n \text{ chẵn}, \\ x^{\left(\frac{n}{2}\right)^2} \cdot x, & n \text{ lẻ}. \end{cases}$$

```
double Pow(double x, int n) {  
    if (n==0) return 1;  
    double t = Pow(x,n/2);  
    return n%2? t*t*x:t*t;  
}
```

Đặt $T(n)$ là độ phức tạp thuật toán:

$$\begin{aligned} T(n) &= C_0 \quad (n = 0, 1) \\ T\left(\frac{n}{2}\right) + C \quad (n > 1) \\ \text{Ta có: } T(n) &= T\left(\frac{n}{2}\right) + C = T\left(\frac{n}{4}\right) + 2C = \dots T(n) = T(n-1) + C = T(n-2) + 2C \\ T(n) &= T\left(\frac{n}{2^k}\right) + kC = C_0 + kC = C_0 + C \cdot \log_2 n = \Theta(\log n) \\ \text{Tính } x^n: & \text{ Khi } n = 0, 1, \quad x^n = 1 \\ & \text{ Khi } n \text{ chẵn}, \quad x^{2^{n/2}} \\ & \text{ Khi } n \text{ lẻ}, \quad x^{2^{n/2}} \cdot x \end{aligned}$$

- o UCLN (suy biến của đệ quy xét theo có cái j return trong hàm)

```
int gcd(int a,int b) {  
    return b? gcd(b,a%b):a;  
}
```

- o Đặt $T(b)$ là độ phức tạp của thuật toán

- Tốt nhất : $T(b) = \omega(1)$
- Xấu nhất : Hai số fibon liên tiếp

- Fibonacci ($T(n) = \theta(\log n)$)

- PHƯƠNG PHÁP CÂY ĐÁNH GIÁ ĐỘ PHỨC TẠP

```
double myFunc(double x, int n) {
    if (n==0) return x;
    double s =x;
    for (int i = 1; i<=n;i+=2) {
        s += (s-i)*(x+i)
    }
    return s + myFunc(x-1,n/2)*myFunc(x+n,n/2);
}
```

Đặt $T(n)$ là độ phức tạp thuật toán:

$$\begin{aligned}
 T(n) &= 0 \quad (n = 0, 1) \\
 T(n) &= T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + nC \quad (n > 1) \\
 \text{Ta có: } 2T\left(\frac{n}{2}\right) + nC &= 2\left(2T\left(\frac{n}{4}\right) + \frac{nC}{2}\right) + nC \\
 &= 2^k \left(T\left(\frac{n}{2^k}\right) + k n C\right) = 2^k T(1) + k n C \\
 &\Rightarrow n C_0 + c n \log n
 \end{aligned}$$

BTVN: $T(n) = 2T(n/3) + cn^3 O(n^3)$

$$T(n) = T(n/2) + T(n/4) + cn^2 O(n^2)$$

- ĐỊNH LÝ MASTER
 - $T(n) = aT(n/b) + Cn^k$
 - Xét ($a = b^k$)
 - Nếu $\log b a > k \Rightarrow T(n) = \theta(n^{\log b a})$
 - Nếu $\log b a < k \Rightarrow T(n) = \theta(n^k)$
 - Nếu $\log b a = k \Rightarrow T(n) = \theta(n^k \log n)$
- VD: $T(n) = T(n/3) + C$
- Áp dụng định lý master (TH2) $\rightarrow a = 1 = b^k \Rightarrow T(n) = \theta(\log n)$

Chương 2 . Tìm kiếm

1.Bài toán tìm kiếm (BFS và DFS)

Cho Không gian trong trạng thái tìm kiếm $K = (S, s, F, R)$ trong đó

- S : tập các trạng thái
- s thuộc S trạng thái khởi đầu
- F thuộc S : tập trạng thái kết thúc
- $R : S \rightarrow S^*$ quy tắc hiệu đổi trạng thái
- Tìm đường đi từ $s \rightarrow f$ thuộc F

2. Thuật toán DFS - Depth First Search (tìm kiếm theo chiều sâu)

a. Ngăn xếp (Stack)

- Khái niệm: là một cấu trúc dữ liệu (động) , tuyến tính, trườ tượng, vào ra cùng một đầu (vào sau ra trước) LIFO
- BT : Cho n a1,a2,a,...an. Với i từ 1→n, với mỗi ai tìm phần tử lớn hơn gần nhất bên trái
- C1: Trực tiếp → $O(n^2)$
- C2: Stack
- **BTVN: Chào đón K59**

3. Tiến lên DFS

- **B1.** Nhập giá trị khởi đầu
 - **n** (ví dụ: đỉnh gốc, số phần tử cần xử lý... tùy bài toán)
- **B2.** Khởi tạo cấu trúc dữ liệu
 - Khai báo **ngăn xếp S** và đẩy **n** vào:
- **B3.** Khởi tạo tập kết quả
 - Khai báo **tập (set) out** rỗng để lưu các giá trị đã thăm/xử lý
- **B4.** Duyệt cho đến khi stack rỗng

```
#include <bits/stdc++.h>
using namespace std;

main() {
    int n;
    cin >> n;
    stack<int> S; S.push(n);
    set<int> Out; Out.insert(n);
    while(not S.empty()) {
        int u = S.top();
        S.pop();
        for (int a = 1; a*a <= u ; a++){
            if (u%a==0) {
                int v = (a-1)*(u/a+1);
                if (Out.find(v) == Out.end()) // Out.count(v)
                {
                    S.push(v);
                    Out.insert(v);
                }
            }
        }
    }
    for (int x : Out) cout << x << " ";
}
```

4. Thuật toán BFS (theo chiều rộng)

a. Hàng đợi queue

- Khái niệm: Hàng đợi là 1 cấu trúc dữ liệu tuyến tính, vào đầu này ra đầu còn lại theo FIFO

→ BT: CÁNH CỬA THẦN KÌ

Hiện nay, trên bảng rank thì ngoài các admin tài năng, Top 5 coder **themis UTC** đang là :

dangdungcntt, tienquanutc, quang123, maianh, nguyenminhduc2820.

Để khích lệ tinh thần chăm chỉ của những sinh viên này, Thầy giáo TICHXP quyết định cho 5 bạn này ước cùng chung một điều ước.

Tất cả đều là sinh viên chăm chỉ nên dành rất nhiều thời gian cho việc học, vì vậy họ đều ế, sau một hồi thảo luận thì 5 bạn đã ước mình sẽ được phân thân ra để có thể vừa học và vừa đi kiếm người yêu, nếu được như vậy cuộc đời họ sẽ trở nên nở hoa :))

Điều ước rất khó thực hiện, nhưng vì Thầy Giáo TICHXP rất tài năng và thương sinh viên, sau 1 thời gian nghiên cứu đã chế tạo ra 1 cánh cửa thần kì, chỉ cần đi vào cánh cửa này là sẽ được phân thân ra 2 người giống nhau.

5 bạn sinh viên sẽ xếp thành 1 hàng để lần lượt đi vào cánh cửa, ban đầu sẽ theo thứ tự rank từ cao xuống lần lượt là :

dangdungcntt, tienquanutc, quang123, maianh, nguyenminhduc2820

Sau mỗi lần ai bước vào cánh cửa thì người đó và bản phân thân của họ sẽ ra đằng sau cùng của hàng chờ, tiếp tục là người kế tiếp.

Ví dụ, sau khi *dangdungcntt* (là người thứ 1) bước qua cánh cửa thì hàng chờ sẽ như sau:

tienquanutc, quang123, maianh, nguyenminhduc2820, dangdungcntt, dangdungcntt.

tiếp đến là *tienquanutc* và *quang123*. Sau khi 2 người này đi qua cánh cửa, hàng chờ sẽ là :

maianh, nguyenminhduc2820, dangdungcntt, dangdungcntt, tienquanutc, tienquanutc, quang123, quang123.

Và cứ tiếp tục như vậy. nhưng vì muốn thử hiệu suất của cánh cửa, Thầy giáo cứ để các bạn thực hiện như vậy cho đến khi cánh cửa hết hiệu lực. Vì vậy hàng chờ sẽ rất dài, bỗng nhiên Thầy giáo tò mò rằng người thứ n bước vào cánh cửa sẽ là ai.

Để có cơ hội nhận được những món quà thú vị từ thầy, các bạn hãy giúp thầy trả lời nhé.

Dữ liệu vào:

- Dòng đầu tiên là t ($1 \leq t \leq 1000$) - số bộ test
- t dòng tiếp theo : mỗi dòng gồm duy nhất số n ($1 \leq n \leq 10^9$)

Dữ liệu ra:

Gồm t dòng:

- Ứng với mỗi test , in ra duy nhất tên của người thứ n sẽ bước vào cánh cửa

INPUT:

```
3
1
8
4534
```

OUTPUT:

```
dangdungcntt
tienquanutc
maianh
```

- Cách làm: Nhập N
 - Khai báo Queue Q đưa vào $(a1) \dots (e1)$
 - Lấy x ptu đầu Q khỏi Q

- $N -= x$ tần suất
- `Q.push(x)`
- Nhân tố ptu cuối
- B4: lặp lại B3 tới $N = \text{ptu đầu}$
- B5: xuất `Q.front()`

```
#include <bits/stdc++.h>
using namespace std;
void sol() {
    int n;
    cin >> n;
    queue<pair<string,int>> Q;
    for (auto x: {"dangdungcntt", "tienquanutc", "quang123", "maianh", "nguyenminhduc2820"}) {
        Q.push({x,1});
    }
    while(n>Q.front().second) {
        n-=Q.front().second;
        Q.push(Q.front());
        Q.pop();
        Q.back().second*=2;
    }
    cout << Q.front().first<<"\n";
}
main() {
    int test;
    cin>>test;
    while(test--) sol();
}
```

→ VD: THANG MÁY

Một tòa nhà cao n tầng được đánh số từ tầng 1 đến tầng n có lắp thang máy. Hiện nay thang máy bị trục trặc do đó mỗi lần có thể di chuyển lên đúng k tầng hoặc di chuyển xuống đúng m tầng nếu có đủ không gian để di chuyển. Thang máy đang ở tầng s một người ở đó muốn di chuyển đến tầng f hỏi số bước di chuyển ít nhất để đến được tầng f , nếu không di chuyển được hãy xuất ra -1 .

• Input

Dòng đầu chứa n là số tầng của tòa nhà ($1 < n < 300$).

Dòng thứ hai chứa hai số k và m tương ứng là số tầng lên và xuống mỗi lần di chuyển của thang máy ($1 \leq k, m \leq 50$).

Dòng thứ ba chứa s và f ($1 \leq s, f \leq n$) là vị trí tầng xuất phát và đích đến.

• Output

Số bước ít nhất di chuyển thang máy, nếu không đến được đích bạn hãy xuất ra -1 .

Example 1

Input

```
12
5 7
1 8
```

Output

11

Giải thích: xuất phát từ tầng 1 mỗi lần lên đúng 5 tầng hoặc xuống đúng 7 tầng nếu có đủ không gian để lên hoặc xuống do đó cần 11 bước chuyển thang qua các tầng $1 \rightarrow 6 \rightarrow 11 \rightarrow 4 \rightarrow 9 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 5 \rightarrow 10 \rightarrow 3 \rightarrow 8$.

Example 2

Input

120
15 57
18 112

Output

-1

Lập bảng (VD)

Bước	Xét	Next	Queue	Map
0	11	18, 7	{18, 7}	$11 \rightarrow 0$
1	18	25, 14	{7, 25, 14}	$18 \rightarrow 1, 7 \rightarrow 1$
2	7	3	{25, 14, 3}	$14 \rightarrow 2$
3	25		{14, 3}	$3 \rightarrow 2$
4	14	21, 10	{3, 21, 10}	$25 \rightarrow 2$
5	3		{21, 10}	ĐÃ ĐẾN TẦNG 3!

THUẬT TOÁN

- **B1:** Nhập n, k, m, s, f
 - n : số tầng
 - k : số tầng có thể **lên**
 - m : số tầng có thể **xuống**
 - s : tầng bắt đầu
 - f : tầng đích
- **B2:**
 - Khai báo hàng đợi Q
 - Đưa s vào Q
 - Khai báo map M để lưu số bước: $M[s] = 0$
- **B3:** Trong khi Q không rỗng thì:
 - Lấy u ra khỏi Q
- **B4:** Xét các bước có thể đi từ u :
 1. Nếu $u + k \leq n$ và $u + k$ chưa có trong M
 - Đưa $u + k$ vào Q
 - $M[u + k] = M[u] + 1$
 2. Nếu $u - m \geq 1$ và $u - m$ chưa có trong M

- Đưa $u - m$ vào Q
- $M[u - m] = M[u] + 1$

• **B5:**

- Nếu f đã nằm trong $M \rightarrow$ Xuất $M[f]$ (số bước tối thiểu)
- Nếu duyệt hết mà không gặp $f \rightarrow -1$

```
#include <bits/stdc++.h>
using namespace std;

main() {
    int n,k,m,s,f;
    cin >> n >> k >> m >> s >> f;
    queue<int> Q; Q.push(s);
    map<int,int> M = {{s,0}};
    while(Q.size() && M.find(f) != M.end()) {
        int u = Q.front();
        Q.pop();
        for (auto x : {k,-m}) {
            if (u+x > 0 && u+x <= n && M.count(u+x) == 0) {
                Q.push(u+x);
                M[u+x] = M[u] + 1;
            }
        }
    }

    if (M.count(f) == 0) {
        cout << -1;
    }
    else {
        cout << M[f];
    }
}
```

→ BT2: ĐONG NƯỚC

Toto bắt đầu học nhập môn trí tuệ nhân tạo với các thuật toán A*. Toto thích thú với bài toán đong nước, thuật toán rất hay nhưng trình độ lập trình còn yếu nên không thể lập trình được. Bạn hãy giúp Toto nhé

Bài toán: Cho hai bình có sức chứa là n và m lít hãy tìm cách đong k lít nước biết rằng mỗi lần đong có thể đổ hết nước trong một bình đi, hoặc đổ thêm nước vào một bình, hoặc đổ nước từ bình nọ sang bình kia và cho số lượng nước để sử dụng là vô hạn và ban đầu hai bình đều chưa chứa nước.

• **Input**

Một dòng gồm 3 số nguyên n, m, k ($1 \leq n, m, k \leq 10000$)

• **Output**

Nếu không đong được xuất ra "Khong dong duoc nuoc", ngược lại chỉ ra số bước ít nhất để đong được k lít nước.

• **Ví dụ 1.**

Input

3 5 4

Output

6

Giải thích : Các trạng thái đóng nước như sau : (0,0)→(0,5)→(3,2)→(0,2)→(2,0)→(2,5)→(3,4)

- Ví dụ 2.

Input

122 24 11

Output

Không đóng được nước

Bước	Xét (x,y)	Next (các trạng thái kế tiếp)	Queue sau bước này	Map cập nhật mới
1	(0,0)	(3,0), (0,5)	{(3,0), (0,5)}	(3,0)→1, (0,5)→1
2	(3,0)	(0,0), (3,5), (0,3)	{(0,5), (3,5), (0,3)}	(3,5)→2, (0,3)→2
3	(0,5)	(3,5), (0,0), (0,0), (3,2)	{(3,5), (0,3), (3,2)}	(3,2)→2
4	(3,5)	(0,5), (3,0), (1,5)	{(0,3), (3,2), (1,5)}	(1,5)→3
5	(0,3)	(3,3), (0,0), (0,0), (3,0)	{(3,2), (1,5), (3,3)}	(3,3)→3
6	(3,2)	(0,2), (3,0), (0,2), (1,4)	{(1,5), (3,3), (0,2), (1,4)}	(0,2)→3, (1,4)→3
7	(1,5)	(0,5), (1,0), (3,3)	{(3,3), (0,2), (1,4), (1,0)}	(1,0)→4
8	(3,3)	(0,3), (3,0), (1,5)	{(0,2), (1,4), (1,0)}	--
9	(0,2)	(3,2), (0,0), (2,0)	{(1,4), (1,0), (2,0)}	(2,0)→4
10	(1,4)	(0,4), (1,0), (0,4)	...	(0,4)→4 ← có 4 lít!

THUẬT TOÁN

- **B1: Nhập** n , m , k
 - n , m : dung tích 2 bình
 - k : số lít nước cần đóng
- **B2: Kiểm tra điều kiện không thể đóng**
 - Nếu $k > \max(n, m)$ hoặc k không chia hết cho $\gcd(n, m)$
→ In "Không đóng được nước" và kết thúc
- **B3: Khai báo hàng đợi** Q và đưa trạng thái ban đầu $(0,0)$ vào Q
- **B4: Khai báo** **Map** để lưu số bước di chuyển đến mỗi trạng thái (x, y)
 - Ban đầu: $\text{Map}[(0,0)] = 0$
- **B5: Lặp khi hàng đợi còn phần tử**
- **B5.1: Lấy** (x, y) từ đầu hàng đợi ra
- **B5.2: Nếu** $x == k$ hoặc $y == k$
→ In $\text{Map}[(x, y)]$ là số bước và kết thúc
- **B5.3: Sinh 6 trạng thái kế tiếp:**

Hành động	Trạng thái mới (nx, ny)
Đổ cạn bình 1	$(0, y)$
Đổ đầy bình 1	(n, y)
Đổ cạn bình 2	$(x, 0)$
Đổ đầy bình 2	(x, m)
Rót từ bình 1 sang bình 2	$(\max(0, x + y - m), \min(x + y, m))$
Rót từ bình 2 sang bình 1	$(\min(x + y, n), \max(0, x + y - n))$

• **B5.4: Với mỗi trạng thái (nx, ny) nếu chưa được xét**

- Thêm vào hàng đợi
- Gán $\text{Map}[(nx, ny)] = \text{Map}[(x, y)] + 1$

• **B6: Nếu duyệt hết mà không gặp trạng thái có k lít nước**

→ In "Khong dong duoc nuoc"

```
#include <bits/stdc++.h>
using namespace std;

typedef pair<int,int> pii;

main() {
    int n,m,k;
    cin >> n>>m>>k;
    queue<pii> Q; Q.push({0,0});
    map<pii,int> M = {{0,0},0};
    while(Q.size()) {
        int x = Q.front().first, y = Q.front().second,t=x+y;
        Q.pop();
        pii Next[] = {{0,y},{n,y},{x,0},{x,m},{max(0,t-m),min(t,m)},{min(t,n),max(0,t-n)}};
        for (auto z: Next) {
            if (M.count(z)==0) {
                Q.push(z);
                M[z] = M[{x, y}] + 1;
                if (z.first==k || z.second==k) {
                    cout << M[z];
                    return 0;
                }
            }
        }
    }
    cout << "\nKhong dong duoc nuoc";
}
```

Chương 3.Quay lui

1. Phương pháp quay lui

- Là phương pháp giải các bài toán đi tìm các cấu hình tổ hợp vector x_1, x_2, \dots, x_k
- G/s có x_1, \dots, x_{k-1}

- TH1: Tới đích → In KQ
- TH2: Duyệt Xk và đệ quy tiếp

→ VD: SINH CÁC DÃY NHỊ PHÂN N

Nhập vào số nguyên dương n ($1 \leq n \leq 9$), hãy sinh các dãy nhị phân có độ dài n .

- **Input**

Số nguyên duy nhất n

- **Output**

Mỗi dòng một dãy nhị phân được liệt kê theo thứ tự từ điển.

- **Example**

Input:

3

Output:

```
000
001
010
011
100
101
110
111
```

```
#include <bits/stdc++.h>
using namespace std;

main() {
    int n ;cin>>n;
    int x[n+5] = {};
    while(x[0]==0) {
        for (int i = 1;i<=n;i++) cout << x[i] << " ";
        cout << "\n";
        int j ;
        for (j = n;x[j]==1;j--) {
            x[j] = 0;
        }
        x[j] = 1;
    }
}
```

2. Vết cặn

→ BT: QUÂN HẬU

Cho một bàn cờ vua có kích thước $n * n$, ta biết rằng quân hậu có thể di chuyển theo chiều ngang, dọc, chéo. Vấn đề đặt ra rằng, có n quân hậu, bạn cần đếm số cách đặt n quân hậu này lên bàn cờ sao cho với 2 quân hậu bất kì, chúng không "ăn" nhau.

- **Input**

Một số nguyên n duy nhất ($n \leq 10$)

- **Output**

Số cách đặt quân hậu.

- **Example**

Input:

4

Output:

2

Ý tưởng: Đặt n con hậu trên bàn cờ $n \times n$, mỗi con nằm ở một hàng khác nhau.

Gọi: $x_i \in \{1, 2, \dots, n\}$

Là vị trí cột của con hậu ở hàng thứ i. Mỗi lời giải là một hoán vị: (x_1, x_2, \dots, x_n)

Điều kiện ràng buộc để các hậu không ăn nhau:

$x_i \neq x_j$ (khác cột)

$|x_i - x_j| \neq |i - j|$ (khác đường chéo)

Giả sử đã có: x_1, x_2, \dots, x_{k-1} Tiếp tục đặt hậu tại hàng k:

- Nếu $k > n$, in ra lời giải: $(1, x_1), (2, x_2), \dots, (n, x_n)$

- Nếu $\leq n$, duyệt các giá trị: $t = 1 \rightarrow n$

- Nếu đặt hậu tại (k, t) không bị các hậu trước ăn:

- Gán $x_k = t$

- Gọi đệ quy với x_1, \dots, x_k

```
#include <bits/stdc++.h>
using namespace std;

int x[1000], n, dem;
map<int, int> A, B, C;

//bool datduoc(int k, int t) {
//    for (int i = 1; i < k; i++) {
//        if (x[i] == t) return 0;
//        if (k - i == abs(t - x[i])) return 0;
//    }
//    return 1;
//}

void TRY(int k) {
    if (k - 1 == n) {
        dem++;
//        for (int i = 1; i <= n; i++) cout << complex<int>(i, x[i]) << (i == n ? "\n": " ");
    }

    for (int t = 1; t <= n; t++) {
```



```

        if (C[t]==0 && A[k-t]==0 && B[k+t]==0) {
            x[k]=t;
            C[t] = A[k-t]=B[k+t]=1;
            TRY(k+1);
            C[t] = A[k-t]=B[k+t]=0;
        }

    }
}

main() {
    cin >> n;
    TRY(1);
    cout << dem;
}

```

3.Quay lui nhánh và cận

→ VD: ĐỔI TIỀN

cho n mệnh giá hãy đổi tiền cho số tờ ít nhất

Ý tưởng: tìm x_1, \dots, x_n số tờ mỗi loại

Sao cho x_i là các số tự nhiên : $x_1a_1 + x_2a_2 + \dots + x_na_n = M / x_1+x_2+\dots+x_n \rightarrow \min$

Giả sử : đã có x_1, \dots, x_{k-1}

Tổng tiền: $x_1a_1 + x_2a_2 + \dots + x_{k-1}a_{k-1} = T \leq M$

Tổng tờ : $x_1 + x_2 + \dots + x_{k-1} = t$

Xét 2 TH :

TH1: $k-1 = n-1$

Nếu $M-T$ chia hết cho a_n thì $res = \min(res, M-T/a_n + t)$

TH2: $k-1 < n-1$: Duyệt $x_k = d \rightarrow x_k + t < res / T + x_k a_k \leq M / R$ thì Quay lại (x1...xk,t+ xA

Ý tưởng: Tìm x_1, x_2, \dots, x_n sao cho:

$$x_1a_1 + x_2a_2 + \dots + x_na_n = M$$

$$x_1 + x_2 + \dots + x_n \rightarrow \min$$

Giả sử đã có: x_1, x_2, \dots, x_{k-1}

$$\text{Tổng tiền: } T = x_1a_1 + x_2a_2 + \dots + x_{k-1}a_{k-1}$$

$$\text{Tổng tờ: } t = x_1 + x_2 + \dots + x_{k-1}$$

Trường hợp 1: Nếu $k = n$ và $M - T$ chia hết cho a_n

$$\text{ta có: } x_n = \frac{M - T}{a_n}, \quad res = \min(res, x_n + t)$$

Trường hợp 2: Nếu $k < n$, duyệt $x_k = d$ sao cho:

$$T + d \cdot a_k \leq M \text{ và } d + t < res$$

Nếu thỏa mãn, đệ quy tiếp với (x_1, \dots, x_k) và tổng tờ mới $t + d$

```

#include <bits/stdc++.h>
#define zuno main
#define ll long long

```

```

#define endl "\n"
#define MOD 1000000007
using namespace std;
int a[101], L[10001];
int zuno() {
    ios_base::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    int n, q;
    cin >> n >> q;
    memset(L, 0, sizeof(L));
    L[0] = 1;
    for(int i=1; i<=n; i++) cin >> a[i];
    for(int i=1; i<=n; i++){
        for(int j=a[i]; j<=10000; j++){
            if(L[j-a[i]]!=0){
                if(L[j]==0) L[j] = L[j-a[i]]+1;
                else L[j] = min(L[j], L[j-a[i]]+1);
            }
        }
    }
    for(int i=1; i<=q; i++){
        int x;
        cin >> x;
        if(L[x]!=0) cout << L[x]-1 << endl;
        else cout << -1 << endl;
    }
    return 0;
}

```

→ BT: CHIA CỦA (VẾT CẠN)

Một ông bố có n tài sản có giá trị lần lượt là a_1, a_2, \dots, a_n . Ông chỉ có hai người con ông muốn đem chia n tài sản này cho 2 người con sao cho chênh lệch giữa 2 người là ít nhất có thể.

- **Input**

Dòng thứ nhất số nguyên dương n là tài sản ($1 \leq n \leq 20$)

Dòng thứ hai chứa n số nguyên dương có giá trị tuyệt đối không quá 10^4

- **Output**

Một số nguyên là độ chênh lệch ít nhất có thể của hai người con

- **Example 1**

Input

```

3
8 4 6

```

Output

```

2

```

Giải thích: Một người nhận 8 một người nhận $4+6=10$ nên chênh lệch ít nhất 2

- **Example 2**

Input

5
10 7 9 6 6

Output

0

Giải thích: Một người nhận $10+9=19$ một người nhận $7+6+6=19$ nên chênh lệch ít nhất 0

Minh họa

cách liệt kê các phương án chia 5 tài sản [47,28,16,28,30] cho hai người con, với:

- Mỗi bit 0/1 tương ứng xem tài sản đó vào nhóm **A** (bit = 0) hay nhóm **B** (bit = 1).
- Cột **A** là tổng giá trị tài sản của nhóm A.
- Cột **B** là tổng giá trị tài sản của nhóm B.
- Cột **Chênh lệch** là $|A-B|$

47	28	16	28	30	A	B	Chênh
0	0	0	0	0	149	0	149
0	0	0	0	1	119	30	89
0	0	0	1	0	121	28	93
0	0	0	1	1	91	58	33
....							

Thuật toán: Xét hai trường hợp:

Trường hợp 1: $k = n$ thì: $res = \min(res, |B - A|)$

Trường hợp 2: $k < n$ thì:

- Nếu $A + a_{k+1} \leq \frac{T}{2}$ thì gọi đệ quy: $TRY(k+1, A + a_{k+1}, B)$
- Nếu $B + a_{k+1} < \frac{T + res}{2}$ thì gọi đệ quy: $TRY(k+1, A, B + a_{k+1})$

```
#include <bits/stdc++.h>
using namespace std;
int n,a[100],T=0,res=1e9;

void TRY(int k,int A,int B) {
    if (res<=1) return;
    if (k==n) res=min(res,B-A);
    else {
        if (A+a[k+1]<=T/2) TRY(k+1,A+a[k+1],B);
        if (B+a[k+1]<(T+res)/2) TRY(k+1,A,B+a[k+1]);
    }
}

main() {
    cin>>n;
    for(int i = 1;i<=n;i++) {cin >> a[i]; T+=a[i];}
    TRY(0,0,0);
    cout<<res;
}
```

Chương 4. Tham lam

1. Phương pháp tham lam

- Phương pháp tham lam giải các bài tập có yếu tố tối ưu mong muốn các tối ưu cục bộ → Tối ưu toàn cục
- Tham lam nhiều khi → xấp xỉ tốt nhất

→ VD: THU HOẠCH CÀ CHUA

Cho n mảnh rừng cà chua, sản lượng a_1, \dots, a_n . Mỗi ngày thu được một mảnh, thì các mảnh sau giảm k đơn vị sản lượng.

Titi là một nông dân có một cánh đồng cà chua gồm n thửa ruộng, gần Tết đến cũng là mùa cà chua chín rộ. Một mình Titi chỉ có thể mỗi ngày thu hoạch cà chua trên một thửa ruộng mà thôi. Tốc độ chín của cà chua quá nhanh nên mỗi ngày thửa ruộng nào chưa thu hoạch thì sản lượng giảm k đơn vị so với ngày hôm trước. Bạn hãy giúp Titi tính toán để thu hoạch cà chua được nhiều nhất nhé.

- **Input**

Dòng thứ nhất số nguyên dương n là số thửa ruộng ($1 \leq n \leq 10^5$) và số nguyên k là độ giảm sản lượng trên mỗi thửa ruộng của từng ngày ($1 \leq k \leq 10^5$)

Dòng tiếp theo chứa n số nguyên dương là sản lượng cà chua ban đầu là các số nguyên không âm có giá trị không vượt quá 109.

- **Output**

Một số nguyên là giá trị sản lượng lớn nhất thu được

- **Example**

Input

```
8 2
4 7 2 8 4 8 3 2
```

Output

```
17
```

- **Bước 1:** Nhập dữ liệu
 - Nhập n là số thửa ruộng, k là độ giảm sản lượng mỗi ngày.
 - Nhập mảng $a[1..n]$ là sản lượng ban đầu của từng thửa ruộng.
- **Bước 2:** Sắp xếp
 - Sắp xếp mảng a theo thứ tự **giảm dần** (tức là thửa có sản lượng nhiều nhất đứng đầu).
 - Mục tiêu: thu hoạch ruộng có nhiều cà chua nhất trước để tránh bị giảm sản lượng.
- **Bước 3:** Thu hoạch theo thứ tự
 - Gọi i là số ngày đã trôi qua (từ 0 đến $n-1$)
 - Với mỗi ruộng ở vị trí i trong mảng đã sắp:
 - Tính sản lượng thực tế: $a[i] - i*k$
 - Nếu sản lượng > 0 → cộng vào kết quả
 - Nếu ≤ 0 → bỏ qua (vì thu hoạch không còn gì)
- **Bước 4:** Xuất kết quả

- In ra tổng sản lượng thu hoạch được sau n ngày.

```
#include<bits/stdc++.h>
using namespace std;
int n,k,res=0;
main() {
    cin>> n>> k;
    int a[n];
    for (int i= 0;i<n;i++) cin>> a[i];

    sort(a,a+n,greater<int>());
    for (int i= 0;i<n;i++) {
        if (a[i]>i*k) res+=a[i]-i*k;
    }
    cout<< res;
}
```

→ BT:LẬP LỊCH

Toto vừa được bổ nhiệm làm chủ tịch của tổng công ty **TNHH_MTV** nên có rất nhiều công việc cần giải quyết. Mỗi công việc có một thời điểm bắt đầu và một thời điểm kết thúc mà Toto không thể làm hai việc được cùng lúc tức là chỉ khi việc này chấm dứt hoàn toàn thì mới bắt đầu được việc khác.

Giả sử bạn là Tổng thư ký cho Toto bạn hãy lập lịch cho Toto sao cho số công việc mà Toto có thể làm là nhiều nhất và không xung đột với nhau.

• Input

Dòng đầu chứa số lượng công việc $n(1 \leq n \leq 105)$.

Tiếp theo n dòng mỗi dòng chứa thời gian bắt đầu và kết thúc của từng công việc $s_i, f_i(1 \leq s_i < f_i \leq 109)$.

• Output

Số công việc được chọn nhiều nhất sao cho các công việc không giao nhau kể cả ở các đầu mút thời gian.

• Ví dụ

Input

```
5
4 7
2 8
3 6
7 9
9 12
```

Output

```
2
```

Minh họa

Công việc	Bắt đầu	Kết thúc	last_end	RES	Có chọn không?	Ghi chú
-----------	---------	----------	----------	-----	----------------	---------

(3,6)	3	6	$-\infty$	0	✓ Có	Cập nhật last_end = 6
(4,7)	4	7	6	+1	✗ Không	Trùng thời gian
(2,8)	2	8	6	0	✗ Không	Trùng thời gian
(7,9)	7	9	6	0	✓ Có	Cập nhật last_end = 9
(9,12)	9	12	9	+1	✗ Không	Giao tại nút
				SUM = 2		

```
#include<bits/stdc++.h>using namespace std;
typedef pair<int,int> event;
#define start first#define end secondboolcmp(event u,event v) {
    return u.end< v.end;
}

main() {
    int n, res= 0, t= -1e9;
    cin>> n;
    event A[n];
    for (auto&a: A) cin>> a.start>> a.end;
    sort(A,A+n,cmp);
    for (auto a:A) if (a.start>t) {
        res++;
        t= a.end;
    }
    cout<< res;
}
```

3.Tham lam và hàng đợi ưu tiên

a. Hàng đợi ưu tiên

- Là một cấu trúc dữ liệu động phi tuyến, trừu tượng trên dữ liệu so sánh được với nhau từng đôi một. Vào thì tùy ý, ra thì ưu tiên
 - push
 - pop
 - top
 - size
 - empty
- Thuật toán Huffman

→ VD: TRÌNH THÁM

Amas là một sinh viên mới vào học tại ĐHGTVT năm thứ nhất. Kết thúc học kỳ 1, Amas được tham gia một học kỳ quân sự. Hôm nay, lớp học của Amas được chia làm 2 phía chiến trường và tổ chức tập trận, Amas tham gia bên phía quân xanh có nhiệm vụ đi trinh thám quân đỏ huấn luyện. Amas leo lên một quả đồi và sử dụng ống nhòm để quan sát thì phát hiện ra quân đỏ đang dàn hàng ngang để tập luyện, khổ nỗi ống nhòm nhỏ nên mỗi khung nhìn chỉ quan sát được một số lính liên tiếp của quân đỏ. Nhiệm vụ trinh thám lần này là tìm ra chiều cao cao nhất của đối phương, để làm được điều đó rất khó nên phải quan sát dần dần lần lượt từ đầu đến cuối của hàng bằng cách dịch khung nhìn lần lượt mỗi lần tịnh tiến lên một đơn vị.

Bài toán đặt ra với Amas là với n quân đồ có chiều lượt là a_1, a_2, \dots, a_n , khung nhìn có khoảng rộng là $1 \leq k \leq n$. Khi dịch khung nhìn lần lượt từ đầu đến cuối thì với từng vị trí khung nhìn phải tìm ra chiều cao của quân đồ cao nhất trong khung nhìn đó

- **Input**

Dòng đầu chứa 2 số n là số quân đồ ($1 \leq n \leq 105$) và k là khung nhìn của ống nhòm ($1 \leq k \leq n$) (số quân đồ tối đa mà một lần ống nhòm quan sát được)

Dòng thứ 2 chứa n số nguyên là chiều cao của quân đồ theo đơn vị chiều cao ($1 \leq a_i \leq 104$)

- **Output**

Kết quả đầu ra gồm $n-k+1$ số nguyên là chiều cao của quân đồ cao nhất trong từng khung nhìn

- **Ví dụ**

- **Input**

```
9 3
7 6 4 2 5 4 3 6 7
```

- **Output**

```
7 6 5 5 4 3
```

C1: Hai vòng for

C2: Minh họa

Bước	Xét	PQ	Out
1	7	{7,1}	
2	6	{7,1},{6,2}	
3	4	{7,1},{6,2},{4,3}	7
4	2	{6,2},{4,3},{2,4}	6
5	5	{4,3},{2,4},{5,5}	5
6	4	{2,4},{5,5},{4,6}	5
7	3	{5,5},{4,6},{3,7}	5
8	1	{4,6},{3,7},{1,8}	4
9	2	{3,7},{1,8},{2,9}	3

- **Bước 1: Chuẩn bị**

- Tạo một **Priority Queue** (hàng đợi ưu tiên - max heap), gọi là **Q**.
 - Mỗi phần tử trong **Q** là một **pair** : (giá trị, vị trí) .

- **Bước 2: Nhập dữ liệu**

- Nhập **n** là số phần tử của mảng.
- Nhập **k** là kích thước cửa sổ trượt.
- Nhập mảng **a[]** gồm **n** số nguyên.

- **Bước 3: Duyệt qua từng phần tử của mảng**

- Với mỗi chỉ số **i** từ **0** đến **n-1** :
 1. Đưa phần tử **a[i]** vào hàng đợi:
 - Đẩy **(a[i], i)** vào **Q** .
 2. Loại bỏ những phần tử không còn trong cửa sổ hiện tại:

→ Trong khi $Q.top().second \leq i - K$, thì $Q.pop()$.

(Vì phần tử đó đã nằm ngoài đoạn $a[i-K+1] \dots a[i]$)

3. In ra giá trị lớn nhất:

- Nếu $i \geq K - 1$ (tức là đã có đủ K phần tử):

→ In $Q.top().first$.

→ VD: PHẦN TỰ TRUNG VỊ

Tichpx dạy môn xử lý ảnh có thuật toán lọc trung vị, đây là một thuật toán cần phải tìm trung vị của histogram. Trung vị của một dãy n phần tử a_1, a_2, \dots, a_n là phần tử mà khi sắp xếp dãy tăng dần thì nó ở giữa dãy đó là $a[(n+1)/2]$. Chẳng hạn trung vị của dãy $\{1, 3, 4, 5\}$ có 4 phần tử trung vị là $a[(4+1)/2] = a[2] = 3$ còn trung vị của dãy $\{1, 3, 4, 5, 5\}$ thì trung vị là $a[(5+1)/2] = a[3] = 4$;

Bài toán đặt ra là bạn lần lượt bổ sung n phần tử vào dãy với mỗi phần tử thứ i bổ sung vào thì xuất ra trung vị của dãy con a_1, a_2, \dots, a_i

• Input

Dòng đầu là số nguyên dương n ($1 \leq n \leq 106$)

Dòng tiếp theo chứa n phần tử có giá trị tuyệt đối không vượt quá 32767.

• Output

Xuất ra n giá trị trung vị của các dãy con tiếp đầu của dãy đã cho

• Ví dụ

Input

```
9
7 4 2 1 6 8 5 8 7
```

Output

```
7 4 4 2 4 4 5 5 6
```

Giải thích : Ban đầu dãy rỗng ta bổ sung lần lượt

- Bổ sung 7 nên dãy con sau khi sắp $\{7\}$ → trung vị 7
- Bổ sung 4 nên dãy con sau khi sắp $\{4, 7\}$ → trung vị 4
- Bổ sung 2 nên dãy con sau khi sắp $\{2, 4, 7\}$ → trung vị 4
- Bổ sung 1 nên dãy con sau khi sắp $\{1, 2, 4, 7\}$ → trung vị 2
- Bổ sung 6 nên dãy con sau khi sắp $\{1, 2, 4, 6, 7\}$ → trung vị 4
- Bổ sung 8 nên dãy con sau khi sắp $\{1, 2, 4, 6, 7, 8\}$ → trung vị 4
- Bổ sung 5 nên dãy con sau khi sắp $\{1, 2, 4, 5, 6, 7, 8\}$ → trung vị 5
- Bổ sung 8 nên dãy con sau khi sắp $\{1, 2, 4, 5, 6, 7, 8, 8\}$ → trung vị 5
- Bổ sung 7 nên dãy con sau khi sắp $\{1, 2, 4, 5, 6, 7, 7, 8, 8\}$ → trung vị 6

Minh họa (i là thêm phải, i chẵn thêm trái)

Bước	Xét	PQ lớn	PQ bé	Trung vị (out)
1	7	{7}	{}	7

2	6	{6}	{7}	6
3	4	{4}	{6, 7}	6
4	2	{4, 2}	{6, 7}	4
5	8	{4, 2}	{6, 7, 8}	6
6	9	{6, 2, 4}	{7, 8, 9}	6
7	7	{6, 2, 4}	{7, 7, 9, 8}	6
8	1	{4, 2, 1}	{6, 7, 7, 9, 8}	4
9	2	{2, 2, 1}	{4, 6, 7, 7, 9, 8}	2

• Bước 1: Khai báo

- Khai báo 2 hàng đợi ưu tiên:
 - **L**: max heap (ưu tiên phần tử lớn nhất) – chứa nửa **nhỏ hơn**
 - **R**: min heap (ưu tiên phần tử nhỏ nhất) – chứa nửa **lớn hơn**

• Bước 2: Nhập dữ liệu

- Nhập số lượng phần tử **n**.

• Bước 3: Duyệt từ $i = 1 \rightarrow n$

Với mỗi phần tử **a[i]** vừa nhập vào:

- ➤ **Nếu i là số lẻ:**
 - Đưa **a[i]** vào **L** (nửa nhỏ hơn).
- ➤ **Nếu i là số chẵn:**
 - Đưa **a[i]** vào **R** (nửa lớn hơn).

• Bước 4: Kiểm tra và hoán đổi

- **Nếu $L.top() > R.top()$** (phần tử lớn nhất bên trái lại lớn hơn phần tử nhỏ nhất bên phải) → **bị sai thứ tự**
 - Lấy **L.top()** và **R.top()** ra, rồi **đổi chỗ cho nhau** để duy trì tính chất:
 - Tất cả phần tử trong **L** \leq tất cả phần tử trong **R**

• Bước 5: In kết quả

- Sau mỗi bước (hoặc khi cần), in **L.top()** → chính là **trung vị hiện tại**

• 🔄 Lặp lại B3 đến khi hết n phần tử.

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    priority_queue<int, vector<int>, less<int>>> L;
    priority_queue<int, vector<int>, greater<int>>> R;
    int n, x;
    scanf("%d", &n);

    for (int i = 1; i <= n; i++) {
        scanf("%d", &x);
        if (i % 2) {
            L.push(x);
        } else {
            R.push(x);
        }
    }
}
```

```

        R.push(x);
    }

    if (i > 1 && L.top() > R.top()) {
        int l_top = L.top();
        int r_top = R.top();
        L.pop();
        R.pop();
        L.push(r_top);
        R.push(l_top);
    }

    printf("%d ", L.top());
}
}

```

→ VD: GIAO HÀNG

Khi đại dịch Covid-19 tràn đến, cả nước áp dụng chính sách cách ly toàn xã hội, mọi người hạn chế ra ngoài. Nhiều người nhận thấy đây là cơ hội kinh doanh tốt và nhanh chóng thành lập công ty vận chuyển hàng hóa giao tận nơi cho khách hàng.

Titi là một nhân viên giao hàng cho một công ty lớn, chính sách của công ty này là cứ giao hàng mà trước một thời hạn cho trước sẽ được thưởng tùy theo món hàng mà nhân viên đó giao.

Titi nhận được n món hàng cần giao, món hàng thứ i giao không trễ quá thời gian t_i sẽ được thưởng v_i tiền. Giả sử cứ một đơn vị thời gian thì Titi giao được một món hàng, bạn hãy giúp Titi xếp thứ tự giao hàng thế nào để được nhiều tiền thưởng nhất nhé

- **Input**

Dòng đầu số nguyên dương n là số món hàng Titi phải giao ($1 \leq n \leq 1000$)

Tiếp theo n dòng mỗi dòng chứa hai giá trị là t_i và v_i nếu món hàng này Titi giao không trễ quá t_i thì sẽ được thưởng v_i trong đó $1 \leq t_i, v_i \leq 106$

- **Output**

Số tiền được thưởng cao nhất có thể nếu Titi chọn được cách giao hợp lý nhất.

- **Ví dụ**

Input

```

6
3 5
3 7
1 3
2 4
2 2
4 1

```

Output

```

17

```

Giải thích Để dễ giải thích ta cứ gọi đơn vị thời gian là giờ. Chúng ta có **6** món đồ chuyển cách chuyển như sau sẽ được **17** tiền:

Giờ thứ nhất chuyển món thứ tư có thời hạn trong 2 giờ nhưng giao luôn giờ thứ nhất bạn được thưởng **4** tiền

Giờ thứ hai và ba chuyển món thứ nhất và thứ hai có thời hạn trong 3 giờ bạn giao một món đúng giờ và một món sớm giờ tổng cộng **12** tiền thưởng

Giờ thứ tư tất nhiên bạn giao món thứ sáu thời hạn trong 4 giờ và bạn vẫn kịp bỏ túi **1** tiền thưởng

THUẬT TOÁN

- **Bước 1: Nhập dữ liệu**

- Nhập n món hàng.
- Với mỗi món a, b (deadline a , phần thưởng b) → đưa vào danh sách $A[a]$.

- **Bước 2: Duyệt từ thời gian $i = 100000$ về 1**

- Với mỗi thời điểm i :
 - Thêm các phần thưởng có deadline là i vào hàng đợi pq .
 - Nếu pq không rỗng → lấy phần thưởng lớn nhất ($pq.top()$), cộng vào res .

- **Bước 3: In tổng thưởng res .**

Minh họa (VD)

Thời gian	Phần thưởng thêm vào PQ	PQ hiện tại	Phần thưởng chọn (top)	Tổng thưởng
4	1	[1]	1	1
3	5, 7	[7,5]	7	8
2	4, 2	[5,4,2]	5	13
1	3	[4,3,2]	4	17

B1: Treo nhỏ

Cột	1	2	3	4	5	6
Hàng 1	1	4	7			1
Hàng 2		2	5			

B2: Hái

Thời gian (cột)	Xét (giá trị treo tại cột)	PQ lớn (sau push)	Pop lấy (out)
6	1	{1}	1
5	–	{ }	–
4	–	{ }	–
3	7, 5	{7, 5}	7
2	4, 2	{5, 4, 2}	5
1	1	{4, 2, 1}	4

- **Giải thích:**

1. Ở cột 6: chỉ có quả 1 → push vào PQ, pop lấy 1.
2. Cột 5 và 4 không có nho → PQ rỗng, bỏ qua.
3. Cột 3: có 7 và 5 → push cả hai, $PQ=\{7,5\}$, pop lấy 7.
4. Cột 2: có 4 và 2 → push cả hai, PQ cũ còn {5}, sau push thành {5,4,2}, pop lấy 5.
5. Cột 1: có 1 → push, $PQ=\{4,2,1\}$, pop lấy 4.

Tổng thu hoạch = $1 + 7 + 5 + 4 = 17$.

```
#include <bits/stdc++.h>
using namespace std;
```

```

int main() {
    list<int> A[100005];
    long long n, res = 0, a, b;
    priority_queue<int> pq;

    scanf("%lld", &n);
    while (n--) {
        scanf("%lld %lld", &a, &b);
        A[a].push_back(b);
    }

    for (int i = 100000; i > 0; i--) {
        for (auto x : A[i]) pq.push(x);

        if (!pq.empty()) {
            res += pq.top();
            pq.pop();
        }
    }

    printf("%lld", res);
    return 0;
}

```

- Thuật toán Prism

→ VD: CÂY KHUNG NHỎ NHẤT

Cho đơn đồ thị liên thông vô hướng có trọng số trên cạnh gồm n đỉnh và m cạnh, các đỉnh được đánh số từ 1 tới n và các cạnh được đánh số từ 1 tới m . Hãy tìm cây khung nhỏ nhất của đồ thị

- **Input**

Dòng đầu chứa hai số n, m ($1 \leq n \leq 10000; 1 \leq m \leq 20000$)

m dòng tiếp theo mỗi dòng chứa ba số nguyên u, v, w . Trong đó (u, v) là chỉ số hai đỉnh đầu mút còn w trọng số của cạnh đó ($1 \leq u, v \leq n; 1 \leq w \leq 1000$)

- **Output**

Ghi ra một số nguyên là trọng số nhỏ nhất của cây khung

- **Ví dụ**



Input

```

7 11
1 2 4
1 3 1
1 7 3
2 4 6
2 3 2
2 5 1
3 5 8
3 6 3

```

4 5 5
5 6 2
5 7 3

Output

14

Minh họa

Bước	Pop từ heap (d,u)	Chọn u?	res sau chọn	Đỉnh thêm vào MST	Heap trước khi push	Push các (w,v) từ u	Heap sau push (top → ...)
1	(0, 1)	có	0	1	–	(4,2),(1,3),(3,7)	(1,3),(3,7),(4,2)
2	(1, 3)	có	1	3	(3,7),(4,2)	(2,2),(8,5),(3,6)	(2,2),(3,6),(3,7),(4,2),(8,5)
3	(2, 2)	có	3	2	(3,6),(3,7),(4,2),(8,5)	(6,4),(1,5)	(1,5),(3,6),(3,7),(4,2),(6,4),(8,5)
4	(1, 5)	có	4	5	(3,6),(3,7),(4,2),(6,4),(8,5)	(3,7),(2,6),(5,4)	(2,6),(3,6),(3,7),(3,7),(4,2),(5,4),(6,4),(8,5)
5	(2, 6)	có	6	6	(3,6),(3,7),(3,7),(4,2),(5,4),(6,4),(8,5)	— (mọi neighbor đã vis hoặc nặng hơn)	(3,6),(3,7),(3,7),(4,2),(5,4),(6,4),(8,5)
6	(3, 6) – skip vì vis	–	6	–	(3,7),(3,7),(4,2),(5,4),(6,4),(8,5)	–	(3,7),(3,7),(4,2),(5,4),(6,4),(8,5)
6'	(3, 7)	có	9	7	(3,7),(4,2),(5,4),(6,4),(8,5)	—	(3,7),(4,2),(5,4),(6,4),(8,5)
7	(3, 7) – skip vì vis	–	9	–	(4,2),(5,4),(6,4),(8,5)	–	(4,2),(5,4),(6,4),(8,5)
8	(4, 2) – skip vì vis	–	9	–	(5,4),(6,4),(8,5)	–	(5,4),(6,4),(8,5)
9	(5, 4)	có	14	4	(6,4),(8,5)	—	(6,4),(8,5)
10	cnt = 7 → dừng	–	14	–	–	–	–

• PRIM

• Khởi tạo

- Chọn đỉnh khởi đầu (ví dụ 1).
- `vis[1..n] = false` (chưa vào MST).
- `minE[1..n] = +∞`, nhưng `minE[1] = 0`.
- Tạo `min-heap` chứa cặp `(minE[v], v)`, khởi đầu có `(0, 1)`.
- `res = 0`, `cnt = 0`.

• Lặp cho đến khi đã lấy đủ `n` đỉnh hoặc heap rỗng:

- Pop `(d, u)` nhỏ nhất từ heap.
- Nếu `vis[u] == true` thì bỏ qua.
- Ngược lại:
 - Đánh dấu `vis[u] = true`,
 - `res += d`,
 - `cnt++`.

- Với mỗi cạnh $(u \rightarrow v, w)$:
 - Nếu $!vis[v]$ và $w < minE[v]$ thì
 - $minE[v] = w$
 - push (w, v) vào heap.
- **Kết quả:** khi $cnt == n$ hoặc heap cạn, res là tổng trọng số MST.

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int,int> pii;
#define ts first
#define dinh second
main() {
    vector<pii> A[10004];
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    map<int,int> L;
    int n,m,res=0;
    cin >> n >> m;
    for (int i = 1 ;i<=m;i++) {
        int u,v,w;
        cin >> u >> v >> w;
        A[u].push_back({w,v});
        A[v].push_back({w,u});
    }

    pq.push({0,1});
    while(pq.size()) {
        pii u = pq.top(); pq.pop();
        if (L[u.dinh]==-1) continue;
        res+= u.ts;
        L[u.dinh] = -1;
        for (auto v : A[u.dinh]) {
            if (L[v.dinh] == 0 || L[v.dinh]> v.ts) {
                L[v.dinh] = v.ts;
                pq.push(v);
            }
        }
    }
    cout << res;
}
```

Chương 5. PTTK Quy hoạch động

1. Phương pháp quy hoạch động

- Phương pháp QHĐ TKTT theo kiểu dưới lên
- Để giải bài toán lớn ta xuất phát từ các bài toán nhỏ, quy hoạch giải các bài toán lớn dần
- Bước sau sử dụng kq của bước trước hoạt động theo biểu thức truy hồi. Dùng vòng lặp để tính từ bé → lớn

→ VD: LÁT GẠCH

- Cho n viên gạch lát sêu $2*n$. Đếm số cách

- VD: $n = 3$ Có 3 cách
- Đặt C_n là số cách leo
- $n = 1 \rightarrow 1$ cách
- $n = 2 \rightarrow 2$ cách
- $n = 3 \rightarrow 3$ cách
- $n = 4 \rightarrow 5$ cách
- $\rightarrow C_n = C_{n-1} + C_{n-2}$

→ VD: LEO THANG

Một chiếc cầu thang có n bậc, một người nào đó có thể bước một bước lên một, hai hoặc ba bậc. Hỏi để lên hết n bậc thang thì người đó có bao nhiêu cách leo thang

- **Input**

Một số nguyên dương $1 \leq n \leq 20$

- **Output**

Một số nguyên dương là số cách lên n bậc thang

- **Example**

Input

4

Output

7

Giải thích: Có thể có các cách lên sau

Cách 1: 1+1+1+1 (Bước từng bậc)

Cách 2: 2+1+1 (Bước đầu 2 bậc sau đó là từng bậc)

Cách 3: 1+2+1

Cách 4: 1+1+2

Cách 5: 1+3

Cách 6: 3+1

Cách 7: 2+2

- Gọi $C(n)$ là số cách để leo lên đúng đến bậc thứ n .
- Tại bước cuối cùng, Titi có thể:
 - Leo lên 1 bậc từ bậc $n-1$,
 - Leo lên 2 bậc từ bậc $n-2$,
 - Leo lên 3 bậc từ bậc $n-3$.
- Do đó ta có mối quan hệ đệ quy: $C(n) = C(n-1) + C(n-2) + C(n-3)$

```
#include <bits/stdc++.h>
using namespace std;
```

```

long step(int n) {
    long C[n+1];
    C[0] = 1;
    C[1] = 1;
    if (n >= 2) C[2] = C[1] + C[0];
    if (n >= 3) C[3] = C[2] + C[1] + C[0];
    for (int i = 4; i <= n; i++) {
        C[i] = C[i-1] + C[i-2] + C[i-3];
    }
    return C[n];
}

main() {
    int n; cin >> n;
    cout << step(n);
}

```

VD: CẮT RUY BĂNG

Hôm nay Ryze tham gia một BigGame, trong đó có trò cắt ruy băng. Ryze phải cắt ruy băng có chiều dài n thành các mảnh thỏa mãn:

- Các mảnh có chiều dài là a hoặc b hoặc c
- Số mảnh cắt được là nhiều nhất

Hãy giúp Ryze xác định số mảnh nhiều nhất có thể cắt được.

Input

Một dòng chứa 4 số nguyên n , a , b , và c ($1 \leq n, a, b, c \leq 4000$)

Output

Đáp án của bài toán. Đề bài đảm bảo luôn tồn tại cách cắt thỏa mãn.

Input:

5 5 3 2

Output:

2

Input:

7 5 5 2

Output:

5 5 3 2

Giải thích:

Test 1: 2 mảnh lần lượt có độ dài là 2 và 3

Test 2: 2 mảnh lần lượt có độ dài là 2 và 5


```
#include <bits/stdc++.h>
using namespace std;

long cut(int n,int a,int b,int c) {
    long C[n+5] = {};
    for (int i = 1;i<=n;i++) {
        C[i] = -1;
        if (i >= a && C[i-a]>=0) C[i] = max(C[i],C[i-a]+1);
        if (i >= b && C[i-b]>=0) C[i] = max(C[i],C[i-b]+1);
        if (i >= c && C[i-c]>=0) C[i] = max(C[i],C[i-c]+1);
    }
    return C[n];
}

main() {
    int n,a,b,c;
    cin >> n >> a >> b >> c;
    cout << cut(n,a,b,c);
}
```

→ VD: SỐ CATALANG

- Số catalan thứ n là cách nhân kết hợp n số hợp với nhau
- Đặt C_n là cách
- $C_1 = 1$
- $C_2 = 1$
- $C_3 = 2$
- $C_4 = 5$ (đặt dấu ngoặc mà không thay đổi vị trí)
- $C_5 = 14$
- $C_n = C_1C_{n-1} + C_2C_{n-2} + C_3C_{n-3} + \dots + C_{n-1}C_1$

→ VD: ĐẾM SỐ XÂU NHỊ PHÂN KHÔNG CHỨA 101

Cho một số nguyên dương n bạn hãy đếm số xâu nhị phân có độ dài n mà không chứa xâu con "101"

- **Input**

Một số tự nhiên duy nhất n ($1 \leq n \leq 106$)

- **Output**

Số xâu nhị phân có độ dài n mà không chứa 101, kết quả có thể rất lớn bạn chỉ cần in ra phần dư của nó chia cho 10^9+7

- **Ví dụ**

Input

4

Output

12

Giải thích Có 12 xâu không chứa xâu con 101 là

0000
0001
0010
0011
0100
0110
0111
1000
1001
1100
1110
1111

Đặt C_n là số dãy nhị phân dài n không chứa 101

gồm A_n kết thúc = 0

gồm B_n kết thúc = 1

Ta có

n	1	2	3	4	5	6
A	1	2	4	7	12	21
B	1	2	3	5	9	16
C	2	4	7	12	21	37

→ $A_n = C_{n-1}$

→ $B_n = B_{n-1} + A_{n-2}$

→ $C_n = 2C_{n-1} + C_{n-2} + C_{n-3}$

```
#include <bits/stdc++.h>
using namespace std;
long mod = 1e9+7;
long find(long n) {
    long A[n+5] = {0,1,2};
    long B[n+5] = {0,1,2};
    long C[n+5] = {0,2,4};
    for (int i = 3; i <= n; i++) {
        A[i] = C[i-1];
        B[i] = (B[i-1] + A[i-2]) % mod;
        C[i] = (A[i] + B[i]) % mod;
    }
    return C[n] >= 0 ? C[n] : mod + C[n];
}

main() {
    long n;
    cin >> n;
    cout << find(n);
}
```

→ VD: TỔNG DÃY CON LIÊN TỤC CÓ TỔNG LỚN NHẤT

Cho một dãy số nguyên $a_1, a_2 \dots a_n$, một dãy con liên tục được sinh ra bằng cách lấy các phần tử liên nhau nào đó trong dãy. Nhiệm vụ của bạn tìm tổng lớn nhất của tất cả các dãy con liên tục sinh ra bởi dãy số nguyên ban đầu.

- **Input**

- Dòng đầu là số nguyên dương N ($1 \leq N \leq 10^5$)
- Dòng 2 là N số nguyên có giá trị ($-1000 \leq a_i \leq 1000$) mỗi số cách nhau bằng ít nhất 1 khoảng trống

- **Output**

- Một số nguyên là tổng lớn nhất

- **Ví dụ 1**

Input

```
7
3 -5 2 -1 4 -6 2
```

Output

```
5
```

Giải thích: tổng lớn nhất là $2 + (-1) + 4$

- **Ví dụ 2**

Input

```
7
3 -5 -2 -1 4 -6 2
```

Output

```
4
```

Minh họa

Dãy:

3 -5 -2 -1 4 -6 2

Vị trí (i)	$a[i]$	current_sum (tại i)	max_sum (tại i)
0	3	$\max(3, 0 + 3) = 3$	$\max(-\infty, 3) = 3$
1	-5	$\max(-5, 3 + (-5)) = -2$	$\max(3, -2) = 3$
2	2	$\max(2, -2 + 2) = 2$	$\max(3, 2) = 3$
3	-1	$\max(-1, 2 + (-1)) = 1$	$\max(3, 1) = 3$
4	4	$\max(4, 1 + 4) = 5$	$\max(3, 5) = 5$
5	-6	$\max(-6, 5 + (-6)) = -1$	$\max(5, -1) = 5$
6	2	$\max(2, -1 + 2) = 2$	$\max(5, 2) = 5$
			→ Dãy: 2 -1 4

```
#include <bits/stdc++.h>
using namespace std;
```

```
long find(long a[], long n) {
    long C[n + 5] = {-1};
```

```

C[0] = a[0];
long res = a[0];
for (long i = 1; i < n; i++) {
    C[i] = max(C[i - 1], 0L) + a[i];
    res = max(res, C[i]);
}
return res;
}

int main() {
    long n;
    cin >> n;
    long a[n + 5];
    for (long i = 0; i < n; i++) {
        cin >> a[i];
    }
    cout << find(a, n);
    return 0;
}

```

→ VD:

BÀI TOÁN ĐỔI TIỀN

Ngân hàng có n mệnh giá tiền có số lượng mỗi tờ tiền là vô hạn gồm a_1, a_2, \dots, a_n . Một người muốn đổi số tiền M theo các mệnh giá tiền này hãy tìm cách đổi sao cho số tờ tiền là ít nhất

- **Input**

Dòng đầu chứa hai số n ($1 \leq n \leq 100$) là số loại mệnh giá ngân hàng có và số truy vấn q ($1 \leq q \leq 1000$)

Dòng tiếp theo chứa n số nguyên dương không vượt quá 104 đôi một khác nhau là các loại mệnh giá tiền mà ngân hàng có
Cuối cùng gồm q dòng mỗi dòng một giá trị M ($1 \leq M \leq 104$) và số tiền muốn đổi

- **Output**

Gồm q dòng mỗi dòng một số nguyên dương là số tờ tiền ít nhất đổi được, trong trường hợp không đổi được tiền thì xuất -1

- **Ví dụ 1**

Input

```

3 2
1 7 5
10
18

```

Output

```

2
4

```

Giải thích : Đổi 10 gồm 2 tờ có mệnh giá 5 còn 18 thì có 4 tờ gồm 5+5+7+1

- **Ví dụ 2**

Input

```

3 2
15 6 8
11
12

```

Ouput

```

-1
2

```

Cách

Đặt C_{ij} là số tờ nhất sử dụng các mệnh giá $a_1, a_2, a_3, \dots, a_i$ đổi số tiền j

Ta có : $C_{i0} = 0$

$C_{0j} = \text{infinity}$

$C_{ij} = C_{i-1j}$ (Nếu $a_i > j$)

$= \min(C_{i-1j}, 1 + C_{ij} - a_i)$ (Nếu $a_i \leq j$)

NOTE: Sao chép đến giá trị mệnh giá, so sánh nhỏ hơn thì lấy cộng dần theo

VD: $C[9] = C[2] + 1 = 3$ (7)

$C[9] = C[6] + 1 = 3$ (3)

C	0	1	2	3	4	5	6
init	0						
5	0					1	
1	0	1	2	3	4	1	2
7	0	1	2	3	4	1	2
3	0	1	2	1	2	1	2

Bước	Mô tả
B1	Nhập n , M , và mảng $a[1..n]$
B2	Khởi tạo mảng $C[n+5][M+5]$ – đây là ma trận DP, với $C[i][j]$ là số phần tử tối thiểu để đạt tổng j khi dùng i phần tử đầu tiên
B3	Duyệt $i = 1 \rightarrow n$, gán $C[i][0] = 0$ (vì tổng 0 thì không cần phần tử nào)
B4	Duyệt $j = 1 \rightarrow M$, gán $C[0][j] = \infty$ (vì không có phần tử nào thì không thể tạo tổng > 0)
B5	Duyệt $i = 1 \rightarrow n$, lồng trong đó là: Duyệt $j = 1 \rightarrow M$:
	Nếu $a[i] > j$: không dùng được phần tử này $\Rightarrow C[i][j] = C[i-1][j]$
	Ngược lại (nếu $a[i] \leq j$) \Rightarrow có 2 lựa chọn: dùng hoặc không dùng: $C[i][j] = \min(C[i-1][j], 1 + C[i][j] - a[i])$
B6	Xuất $C[n][M]$ – số phần tử ít nhất cần có để tạo thành tổng M

```

//CO TRUY VAN TRACE
#include <bits/stdc++.h>
using namespace std;

int C[100][1000] = {},n,M,a[100];

void TRACE(int n,int M) {
    if (C[n][M]==0) return;

```

```

while(C[n][M]==C[n-1][M]) n--;
TRACE(n,M-a[n]);
cout<<a[n]<<" ";
}

main() {
    cin >> n >> M;
    for(int i = 1; i<=n;i++) cin >> a[i];
    fill(C[0]+1,C[0]+M+1,1e9);
    for (int i = 1;i<=n;i++) {
        for (int j = 1;j<=M;j++) {
            if (a[i]>j) C[i][j] = C[i-1][j];
            else C[i][j] = min(C[i-1][j],1 + C[i][j-a[i]]);
        }
    }
    if (C[n][M]==1e9) cout << "khong doi duoc";
    else { cout << C[n][M] << "\n";TRACE(n,M);}
}

```

```

//DANH CHO LAP TRINH ONLINE
#include <bits/stdc++.h>
using namespace std;

```

```

const int MAX_M = 40005;
int C[MAX_M] = {}, n, q;

```

```

main()
{
    cin>>n>>q;
    fill(C + 1, C + MAX_M, 1e9);
    C[0] = 0;
    for(int i=1;i<=n;i++){
        int a;
        cin>>a;
        for(int j=a;j<=MAX_M;j++)
            C[j]=min(C[j], 1+C[j-a]);
    }
    while(q--){
        int M;
        cin>>M;
        if (C[M] == 1e9) C[M] = -1;
        cout << C[M] << "\n";
    }
}

```

→ VD: SẮP XẾP BALO KHÔNG LẶP

Cho n đồ vật, mỗi đồ vật có kích thước và giá trị lần lượt là $\langle w_1, v_1 \rangle, \langle w_2, v_2 \rangle, \dots, \langle w_n, v_n \rangle$, một ba lô có kích thước M , hãy tìm cách xếp các đồ vật vào ba lô sao cho tổng kích thước các đồ vật không vượt quá kích thước của ba lô, mà tổng giá trị thu được là lớn nhất.

Giả sử rằng hình dạng của đồ vật và ba lô không ảnh hưởng tới việc xếp đồ, tức là cứ xếp thoải mái miễn là tổng kích thước các đồ vật xếp vào không vượt quá kích thước ba lô là được và chú ý rằng mỗi đồ vật chỉ có số lượng là 1 thôi nhé

- **Input**

Dòng đầu chứa số đồ vật $n(1 \leq n \leq 100)$

Tiếp theo n dòng mỗi dòng chứa hai giá trị $\langle w_i, v_i \rangle$ tương ứng là kích thước và giá trị các đồ vật ($1 \leq w_i, v_i \leq 106$)

Tiếp theo chứa số lần truy vấn $q(1 \leq q \leq 10000)$

Tiếp theo là q dòng mỗi dòng tương ứng với một kích thước ba lô cần truy vấn $M(1 \leq M \leq 104)$

- **Output**

Gồm q dòng mỗi dòng tương ứng với tổng giá trị lớn nhất theo từng truy vấn

- **Ví dụ**

Input

```
6
4 7
3 5
4 8
2 3
2 4
5 9
4
11
100
1
8
```

Output

```
21
36
0
15
```

Giải thích

test 1: $M=11$ thì ta chọn đồ vật $\langle 4,8 \rangle, \langle 2,4 \rangle, \langle 5,9 \rangle$ tổng kích thước $4+2+5=11$ tổng giá trị là $8+4+9=21$

test 2: $M=100$ thì ta chọn tất cả các đồ vật tổng kích thước $4+3+4+2+2+5=20$ tổng giá trị là $7+5+8+3+4+9=36$

test 3: $M=1$ thì không chọn được đồ vật nào nên tổng kích thước là 0 và tổng giá trị là 0

test 14: $M=8$ thì ta chọn đồ vật $\langle 4,8 \rangle, \langle 4,7 \rangle$ tổng kích thước $4+4=8$ tổng giá trị là

- 🧠 **Ý tưởng chi tiết:**

- Gọi $C[i][j]$ là **giá trị lớn nhất** có thể đạt được khi:

- Xét i món đồ đầu tiên
- Với ba lô có dung lượng là j


- 📌 **Công thức chuyển:**

- Nếu không xét món đồ thứ i :

$$C[i][j] = C[i-1][j]$$

- Nếu xét món đồ thứ i và **kích thước món đồ** $a[i] \leq j$:

$$C[i][j] = \max(C[i-1][j], C[i-1][j - a[i]] + b[i])$$

-  **Khởi tạo:**
- $C[0][j] = 0$ với mọi j (không có món nào thì không có giá trị nào)
- $C[i][0] = 0$ với mọi i (ba lô không có sức chứa thì không chứa gì được)

Minh họa

GT KT	0	1	2	3	4	5	6
init							
7 4	0	0	0	0	7	7	7
3 2			3	3	7	7	10
11 5			3	3	7	11	11
8 4			3	3	8	11	11
4 2			4	4	8	11	12
1 1		1	4	5	8	11	12
5 3		1	4	5	8	11	12
1 3		1	4	6	8	11	12

```
//CODE TLE KHI OLP
#include <bits/stdc++.h>
using namespace std;

int C[100][1000] = {},n,a[100],b[100],q;

void TRACE(int n,int M) {
    while (n > 0 && M > 0) {
        if (a[n] <= M && C[n][M] == C[n - 1][M - a[n]] + b[n]) {
            cout << "Chon vat " << n << " kt: " << a[n] << " gt: " << b[n] << "\n";
            M -= a[n];
        }
        n--;
    }
}

main() {
    cin >> n;
    for(int i = 1; i<=n;i++) cin >> a[i] >> b[i];

    cin >> q;
    while(q--) {
        int M;
        cin >> M;
        fill(C[0]+1,C[0]+M+1,1e-9);
        for (int i = 1;i<=n;i++) {
            for (int j = 0;j<=M;j++) {
                if (a[i]>j) C[i][j] = C[i-1][j];
                else C[i][j] = max(C[i-1][j], b[i] + C[i-1][j - a[i]]);
            }
        }
        if (C[n][M]==1e-9) cout << -1 << endl;
        else { cout << C[n][M] << "\n";TRACE(n,M);}
    }
}
```



```

    }

}

//CODE SUBMIT OLP
#include <bits/stdc++.h>
using namespace std;
const int MAX_M = 30005;
int C[MAX_M];
int a[105], b[105], n, q;
vector<int> queries;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> n;
    for(int i = 1; i <= n; i++) cin >> a[i] >> b[i];

    cin >> q;
    queries.resize(q);
    int maxM = 0;
    for(int i = 0; i < q; i++) {
        cin >> queries[i];
        maxM = max(maxM, queries[i]);
    }

    fill(C, C + maxM + 1, 0);
    for (int i = 1; i <= n; i++) {
        for (int j = maxM; j >= a[i]; j--) {
            C[j] = max(C[j], C[j - a[i]] + b[i]);
        }
    }

    for (int i = 0; i < q; i++) {
        cout << C[queries[i]] << "\n";
    }

    return 0;
}

```

VD: XÂU CON CHUNG DÀI NHẤT

Bài toán: Cho chuỗi x và chuỗi y một chuỗi con chung của x và y thu được bằng cách xóa đi 1 số ký tự nào đó trong x và trong y phần còn lại giữ nguyên thứ tự. Cần tìm tất cả các chuỗi con chung dài nhất của x và y .

- **Input**

Nhập vào hai chuỗi ký tự trên hai dòng mỗi chuỗi không có độ dài từ 1 đến 100

- **Output**

Tất cả các chuỗi con chung dài nhất của hai chuỗi trên được viết trên từng dòng theo thứ tự từ điển

Trong trường hợp không có chuỗi con chung xuất ra "khong co xau con chung"

- **Ví dụ 1**

Input

abacab
cavadbdasf

Output

aaa
aab
aba
cab

• Ví dụ 2

Input

concua
embe

Output

khong co xau con chung

Công thức truy hồi:

• Trường hợp cơ sở:

- Nếu một trong hai chuỗi rỗng:

$$C[i][0]=0$$

$$C[0][j]=0$$

• Trường hợp thông thường:

- Nếu $x[i] == y[j]$:

$$C[i][j]=C[i-1][j-1]+1$$

- Ngược lại:

$$C[i][j]=\max(C[i-1][j], C[i][j-1])$$

• 📌 Giải thích:

- Nếu 2 ký tự cuối giống nhau ($x_i = y_j$), thì ta cộng thêm 1 vào LCS của phần còn lại ($x_1...x_{i-1}$, $y_1...y_{j-1}$)
- Nếu khác nhau, thì ta chọn xem bỏ ký tự nào sẽ cho kết quả tốt hơn:
 - Bỏ x_i → xét $C[i-1][j]$
 - Bỏ y_j → xét $C[i][j-1]$

C		X	G	G	A	X	T
	0	0	0	0	0	0	0
A	0	0	0	0	1	1	1
T	0	0	1	1	1	1	2
G	0	0	1	1	1	1	2
X	0	1	1	1	1	2	2

A	0	1	1	1	2	2	2
G	0	1	2	2	2	2	2
T	0	1	2	2	2	2	3

THUẬT TOÁN

• Bước 1: Nhập input

- Nhập 2 chuỗi `x` và `y`
- Gắn thêm ký tự đặc biệt `$` vào đầu mỗi chuỗi để xử lý chỉ số từ 1 (xử lý thuận tiện hơn)
- Đặt `n = độ dài x`, `m = độ dài y`

• Bước 2: Khởi tạo bảng động

- Khai báo mảng `C[n+1][m+1]` để lưu độ dài LCS giữa các tiền tố của `x` và `y`
- Khởi tạo:
 - `C[i][0] = 0` với mọi `i`
 - `C[0][j] = 0` với mọi `j`

• Bước 3: Quy hoạch động

- Duyệt `i = 1 → n`, `j = 1 → m`
- Với mỗi cặp `(i, j)`:
 - Nếu `x[i] == y[j]`:
 - `C[i][j] = 1 + C[i-1][j-1]`
 - Ngược lại:
 - `C[i][j] = max(C[i-1][j], C[i][j-1])`

• Bước 4: Độ dài LCS

- Kết quả độ dài xâu con chung dài nhất là `C[n][m]`

• Bước 5: Truy vết

- Duyệt ngược từ `C[n][m]` về `C[0][0]` để tìm ra một xâu con chung
 - Nếu `x[i] == y[j]`, đưa ký tự đó vào kết quả và đi về `C[i-1][j-1]`
 - Nếu không, đi theo hướng có giá trị bằng `C[i][j]`:
 - Nếu `C[i-1][j] == C[i][j]`: đi lên
 - Nếu `C[i][j-1] == C[i][j]`: đi trái

```
#include<bits/stdc++.h>
using namespace std;
int C[105][105] = {},n,m;
string x,y;
void TRACE(int n,int m){
    if(C[n][m] == 0) return;
    while(C[n][m] == C[n-1][m]) n--;
    while(C[n][m] == C[n][m-1]) m--;
    TRACE(n-1,m-1);
    cout << x[n];
}
int main(){
```

```

cin >> x; n= x.size(); x = "$" + x;
cin >> y; m = y.size(); y = "$" + y;
for(int i=1;i<=n;i++)
for(int j=1;j<=m;j++) C[i][j] = x[i] == y[j] ? 1+C[i-1][j-1] : max(C[i-1][j],C[i][j-1]);
cout << C[n][m] << "\n";
TRACE(n,m);
}

```

```

//Tichpx- tim tat ca cac xccd
//LAPTRINHONLINE
#include<bits/stdc++.h>
using namespace std;
string x,y;
int C[105][105]={},n,m;
set<string> P[105][105];
int main()
{
    cin>>x; n=x.size(); x="$"+x;
    cin>>y; m=y.size(); y="$"+y;
    for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
        if(x[i]==y[j])
        {
            C[i][j]=C[i-1][j-1]+1;
            if(C[i][j]==1) P[i][j].insert(string(1,x[i]));
            else for(auto s:P[i-1][j-1]) P[i][j].insert(s+x[i]);
        }
        else
        {
            C[i][j]=max(C[i-1][j],C[i][j-1]);
            if(C[i-1][j]==C[i][j]) for(auto s:P[i-1][j]) P[i][j].insert(s);
            if(C[i][j-1]==C[i][j]) for(auto s:P[i][j-1]) P[i][j].insert(s);
        }

    for(auto s:P[n][m]) cout<<s<<"\n";
}

```

LƯU Ý KHI ĐI THI:

GK - 1 CONTEST 1h để lấy điểm quá trình

CK - Thi tự luận gồm 3 phần độ phức tạp, phân tích thuật toán (minh họa, thuật toán, code)